Assignment No 3

Anand Uday Gokhale

10th February 2019

1 Abstract

This week's Python assignment will focus on the following topics:

- Reading data from files and parsing them
- Analysing the data to extract information
- Study the effect of noise on the fitting process
- Plotting graphs

2 Introduction

This weeks assignment starts off with generating data as a linear combination of the Bessel Function and $y = x$. Varying amounts of Noise is added to this combination.

$$f(t) = A * J_2(t) - B * t + n(t)$$

Where :

- $A = 1.05$
- $B = 0.105$
- $n(t) =$ noise function
- $J_2(t) =$ Bessel function

We have to study the relation between the error of our estimation of A and B and the standard deviation of the noise that was added.

3 Assignment Problems

3.1 Part 1 : Generation Of Data

Data is generated using the script provided as a part of the assignment. It utilizes the pylab library to find the value of the Bessel function and different values of t and adds noise for 9 values of standard deviation.

3.2 Part 2 : Importing the Data

Numpy's loadtxt function was used to import data from the file where it was previously stored. The data consists of 10 columns. The first column is time, while the remaining columns are data with varying noise levels.

```
def load(FILENAME):
    data = np.loadtxt(FILENAME)
    x = data[:,0]
    y = data[:,1:]
    return x,y
```

3.3 Part 3 : Plotting the Data

This is done using Matplotlib's pyplot Library

```
def plot_with_legend(x,y):
    plt.plot(x,y)
    scl=np.logspace(-1,-3,9)
    plt.xlabel(r'$t$',size=20)
    plt.ylabel(r'$f(t)+n$',size=20)
    plt.legend(scl)
    plt.show()
```

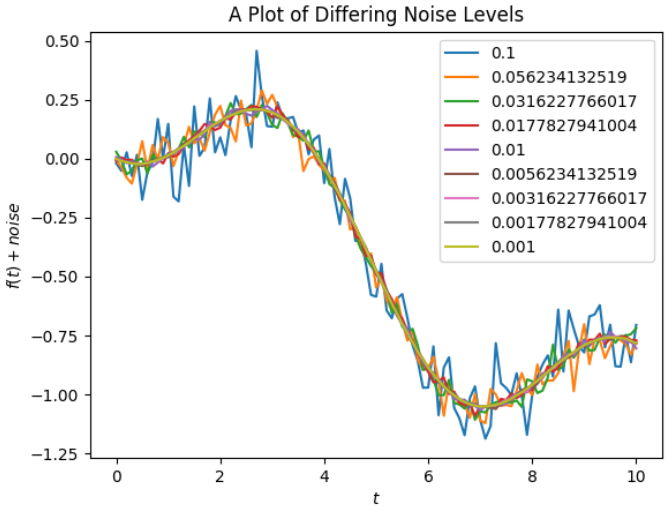

Figure 1: Part 3 : Varying Noise Levels

3.4 Part 4: Plotting the Original function

With $A = 1.05$ and $B = -0.105$ the original function is plotted as follows:

```
def g(t,A=1.05,B=-0.105):
    return A*sp.jn(2,t)+B*t
def plot_g(t):
    plt.figure(0)
    plt.plot(x,g(x))
    plt.show()
```

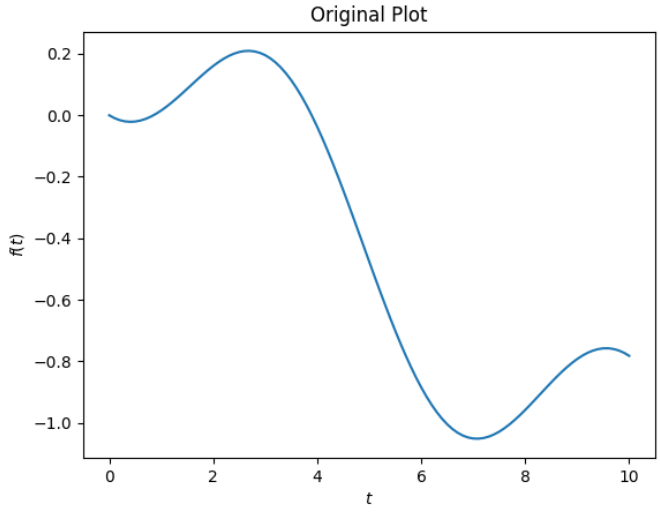

Figure 2: The Original function

3.5 Part 5: Plotting Error Bars

```
def plot_errorbar(x,y,i):
    y_true = g(x)
    sigma = np.std(y[:,i]-y_true)
    plt.plot(x,y_true)
    plt.errorbar(x[::5],y[::5,i],sigma,fmt='ro')
    plt.show()
```

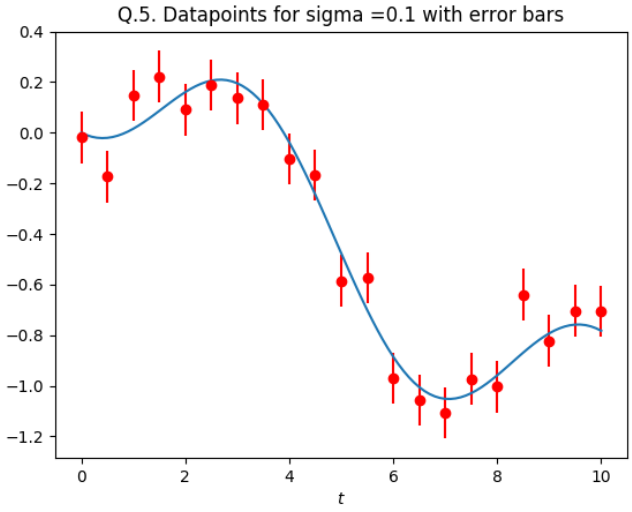

Figure 3: Part 5: Error bars

3.6 Part 6: Generating Matrix

To generate the Matrix M:

$$M = \begin{bmatrix} J_2(t_1) & t_1 \\ J_2(t_2) & t_2 \\ \dots & \dots \\ J_2(t_m) & t_m \end{bmatrix}$$

```
def generateM(x):
    M = np.zeros((x.shape[0],2))
    M[:,0] = sp.jn(2,x)
    M[:,1] = x
    return M
```

To Find Mean Square Error Between the function generated and the original function:

```
def error(x,AB):
    M = generateM(x)
    y_true = np.reshape(g(x),(101,1))
    y_pred = np.matmul(M,AB)
    return (np.square(y_pred - y_true)).mean()
```

3.7 Part 7: Generating Error For Different Values of A and B

We can find the MSE(Mean square error) between our predicted AB vs Actual AB using this formula:

$$\epsilon_{ij} = \sum_{k=0}^{101} (f_k - g(t_k, A_i, B_j))^2 \tag{1}$$

where $f_k = (k+1)^{th}$ column of the data

```
def generateAB(i,j,step1 = 0.1,step2 = 0.01,Amin=0,Bmin = -0.2):
    AB = np.zeros((2,1))
    AB[0][0] = Amin +  step1 * i
    AB[1][0] = Bmin +step2 * j
    return AB
def find_error_matrix(x,y,noise_index):
    try:
        y_noisy = np.reshape(y[:,noise_index],(101,1))
    except:
        y_noisy =np.reshape(g(x),(101,1))
    error = np.zeros((21,21))
    M = generateM(x)
    for i in range(21):
        for j in range(21):
            error[i,j] = np.square( np.matmul(M,generateAB(i,j)) - y_noisy).mean()
    return error
```

3.8 Part 8: Plotting Contour of Error

```
def plot_contour(x,y):
    a = np.linspace(0,2,21)
    b = np.linspace(-0.2,0,21)
    X, Y = np.meshgrid(a,b)
    error = find_error_matrix(x,y,0)
    CS = plt.contour(X,Y,error)
    plt.clabel(CS, inline=1, fontsize=10)
    plt.show()
    return
```

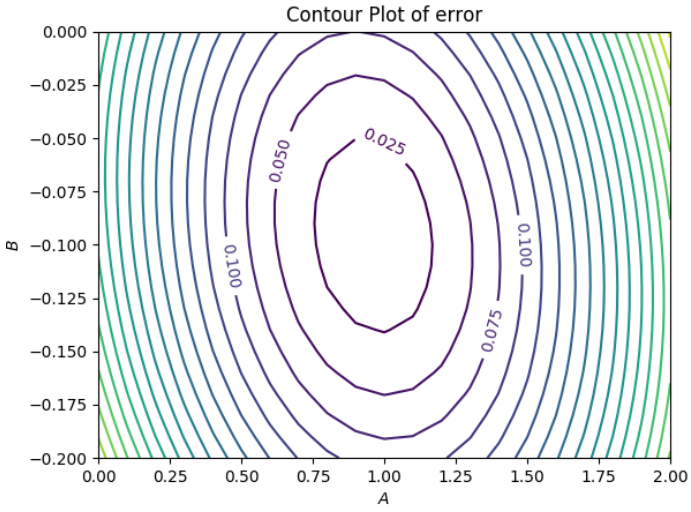

Figure 4: Part 7: Contour Plot

3.9 Part 9 : Estimating A and B

AB is estimated by minimizing $|M * (AB) - b|$
where b = one of the columns of the data
This can be done using the scipy.linalg.lstsq function

```
def estimateAB(M,b):
    return scipy.linalg.lstsq(M,b)
prediction,_,_,_ = estimateAB(generateM(x),y[:,1])
```

3.10 Part 10 : Error of A and B on a linear Scale

```
plt.plot(scl,error_a,'r—')
plt.scatter(scl,error_a)
plt.plot(scl,error_b, 'b—')
plt.scatter(scl,error_b)
plt.legend(["A","B"])
plt.show()
```

3.11 Part 11 : Error of A and B on a loglog Scale

Plotting this is similar to the previous part except for using the function plt.loglog instead of plt.plot.There is also an additional parameter to this function. i.e. $basex$. For convenience I have used $basex = 10$.

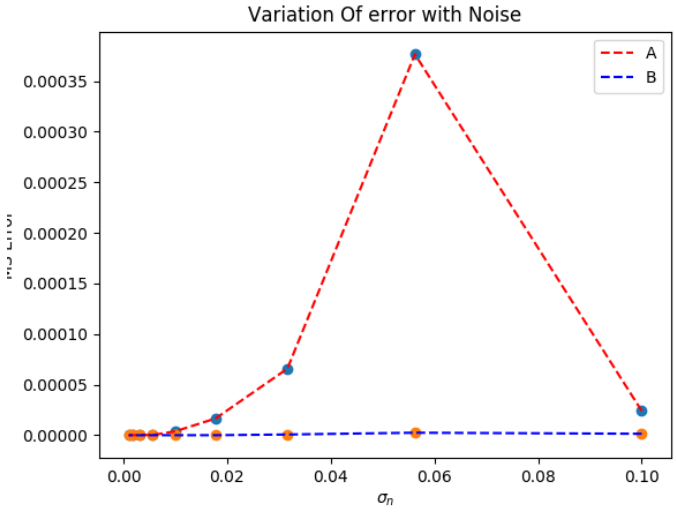

Figure 5: Error of A and B on a linear Scale

This graph is not Linear. However when we plot the error returned by the lstsq function It is nearly Linear This is because our noise varies on a logarithmic scale and it is expected that the error must also vary in a logarithmic manner because the error in the estimation is also based on a Gaussian distribution.

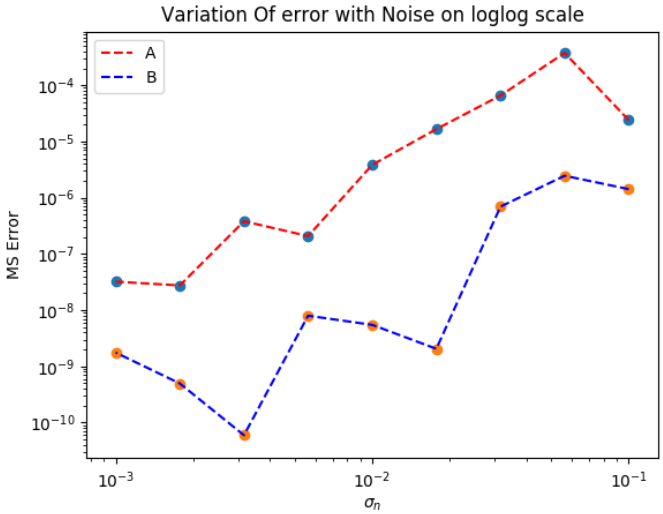

Figure 6: Error of A and B on a LogLog Scale

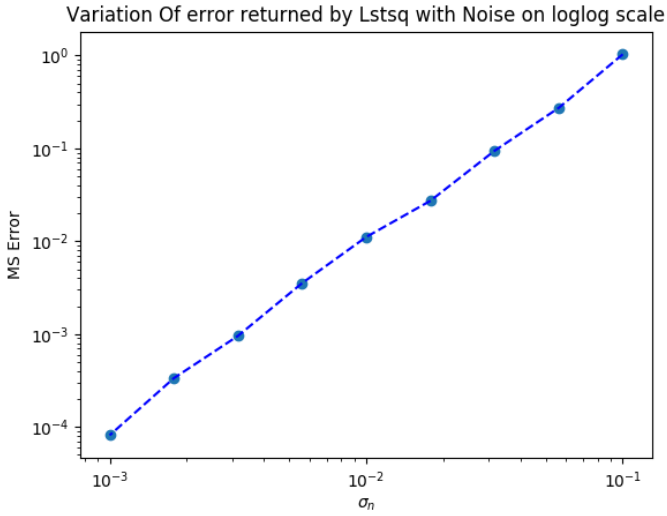

Figure 7: Error returned by lstsq on a LogLog Scale

4 Conclusion

We have used the scipy library to estimate coefficents of a linear combination of 2 functions with varying amounts of noise. As the noise in the data used to estimate the linear combination increases, the error of the prediction also increases on a linearly on a loglog scale.